Load Balancing Grid Scheduler for the Computational Grid Environment

Mr.R.Rajkumar1, Dr.S.Thamarai Selvi2, Mr.G.Kannan3

1Student, 2Professor and Principle Investigator, 3Research Associate

Department of IT, MIT Campus, Anna University, Chennai, India

Abstract - Distributed computing paradigm is a wide spread technology, where computational grid provides huge computation power for the large scale distributed application. One of the challenging issue in computational grid is load balancing. This paper, proposed a new load balancing scheduler algorithm which can not only increases the utilization of the resource and throughput, but also realize the load balancing within the grid environment. The updated topology and load information is acquired dynamically from the resource using the event notification approach. In order to maximize the utilization of resources and to increase the performance of the system application level load balancing is needed for the individual parallel jobs. In many approaches load balancing is done only at the local scheduler level, which is applicable to small application and leads to more communication overhead between the nodes. For the large scale application load balancing at the local scheduler level will not provide the feasible solution. So the novel load balancing algorithm is proposed, which provides the load balancing at the meta-scheduler level. To initiate the load balancing triggering policy is used, which determines the appropriate time period time to start the load balancing operation. Triggering policy can be initiated by using two approaches such as threshold and boundary value app oach. These approach increases the performance in the large scale application by submitting the job to the least loaded machine to reduce the elapsed time and waiting time of job, and to maximize the utilization of the resources which are idle or least loaded.

Index Terms – Load balancing, topology information, load information, event notification, triggering policy, threshold value and boundary value approach.

1 INTRODUCTION

A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. One of the challenging issue in grid environment is Load Balancing, which makes the coordination and scheduling of the resource dynamically to execute the tasks more quickly [1], [14]. Grid technology provides the best toolkit and the system platform to deal with these issues. A vast array of Globus Toolkit (GT4) middleware has been developed to support applications in Grid [2], [15].

The impact of using Load Balancing Scheduler (LBS) algorithm is to choose the best fit resource from the Grid Environment, which executes the task more quickly than the other resource. In the grid environment, each cluster is considered as a resource [19]. There are many Grid Scheduling algorithms exist, in which the resource with high CPU speed and free memory is selected as the best resource for task execution. These types of algorithm may lead to overhead in the case, where the newly arrived task requires more CPU speed and free memory. This situation leads to overloading of the particular resource due to demand and some times, the chance for the unavailability of the resource.

So far the load balancing algorithm is implemented by exploiting the static or dynamic resource information, topology information and load information of the resource, such as loaded or unloaded [12], [13]. They have exploited the information as a separate model, but they didn't integrate all the information in the single algorithm. In order to avoid overloading and unavailability of the resource we have proposed a

new LBS algorithm by exploiting both topology and load information, which is obtained dynamically from the Grid resource.

Of course there are many load balancing algorithms, which act more efficiently in grid environment. Since the jobs are dynamically submitted to scheduler, there may be a possibility to have same requirement for many jobs. As a result Grid scheduler will submit more number of jobs to the single resource, which leads to overloading. In case, if the local users fire the job to the resource without the knowledge of meta-scheduler, then the resource is forced to handle too many number of jobs from both meta-scheduler as well as local user and this situation leads to overloading of resource.

In order to avoid the overloading and job waiting time, load in the particular resource is needs to be balanced. So the Load Balancing is a preferable solution to provide application level load balancing for individual jobs [9]. The impact of using LBS algorithms in the meta-scheduler will reduce the total elapse time, waiting time of the job and maximize the utilization of the resource which is idle or least loaded [9], [17], [19].

2 RELATED WORK

The most important issues in grid environment, is the performance degradation caused due to load imbalance and achieving minimum response time for the client job request [16]. Therefore Load Balancing is indispensable for the heterogeneous cluster, to assure fine distribution of workload on each cluster. All these approaches, broadly implements load sharing algorithms, which can be static or dynamic and also uses the centralized or distributed control [3]. The reference shows that a hybrid of both static and dynamic strategy for the resource selection provides various Load Balancing policies for providing a good performance [13].

In the literature dynamic load balancing technique for grid application based on Graph partitioning, which exploits knowledge of the topology of the Grid environment to partition the communication graph in such a way as to reduce the cross-site communication [12]. For the dynamic load balancing algorithm, it is unacceptable to frequently exchange state information because of high communication overhead. In [9], sender processor collects the status information about neighboring processors by communicating with them at every load balancing instant. For the large scale Grid Environment where communication latency is very large, the status exchange at each load balancing instant can leads to large communication overhead [9], [12].

In our approach the problem of frequent exchange of information is alleviated by estimating the load information on demand by using the event notification approach. Here the triggering policy is considered which makes the decision at what time the load balancing operation is to be initiated [3].

3 DYNAMIC LOAD BALANCING GRID SCHEDULER

3.1 Load Balancing policy

In order to maximize the utilization of resource and to reduce the waiting time of the job, application level load balancing is needed for individual parallel jobs. As per the grid environment is concern, load balancing is to be done by considering any one of the load balancing policies such as transfer policy, selection policy, location policy, information policy and triggering policy [3], [13].

In our proposed model we have considered only four policies. First is the transfer policy, which is also called as threshold policy and the thresholds are expressed in units of load [7], [28]. Suppose new job or task originates at the node, when the load at the node exceeds a threshold T, the transfer policy decides that the node is a sender. In case the node falls below threshold T, then the transfer policy decides that the

particular node can be a receiver for the remote task. Second is the location policy, the responsibility of this policy is to find suitable nodes to share load. A widely used method for finding a suitable node is through polling. In polling, a node polls another node to find out whether it is a suitable node for load sharing. Nodes can be polled either serially or parallel. An alternative to polling is to broadcast a query to find out if any node is available for load sharing. Third is the information policy, which is responsible for making decision such as when information about the states of other nodes in the system should be collected, where it should be collected from, and what information should be collected. Of course there are many information policies available, but we have considered the sender initiated demand-driven policy. In the sender initiated policy, sender will look for the receiver to transfer their loads, and the reverse one is receiver initiated policy [13].

Finally we considered the triggering policy, which determines the appropriate period to start a load balancing operation. The triggering policy can be initiated by any one of the two approaches. In the threshold approach, threshold value is set for the resource or cluster load present in the grid environment. If the resource load exceeds the particular threshold value, then the load balancing policies such as information, selection and location policy are considered, to migrate the job to other resources which are below the threshold value [8]. The boundary value approach is used, if no job arrives to the meta-scheduler for certain time interval, then the average load of the resources is calculated. The upper and lower boundary value is set for the resource, suppose if the resource load exceeds the upper boundary value means then the load balancer will migrate the job to the resource which is below the lower boundary value [5]. Here the job migration takes place until the resource comes to moderate load.

3.2 Load Balancing Grid Scheduler model

In this paper, Load Balancing Grid Scheduler (LBGS) model is proposed using Load Balancing Scheduler (LBS), Load Balancer (LB) and Job Migration (JM) algorithms. This scheduler model is shown in Fig 1. The users will submit their jobs to the meta-scheduler which falls in the queue of request handler. The Request Handler service provides a user interface through which a client can submit the jobs described using JSDL specification to the underlying meta-scheduler. This service will obtain the jobs from user and stores it in a queue of request handler. The dispatch manager, which is present in the CARE Resource Broker (CRB) obtains the submitted job information periodically from the queue, which is implemented in the request handler component [4]. It sends the jobs to LBS for discovering suitable resources for every scheduling interval it maintains.

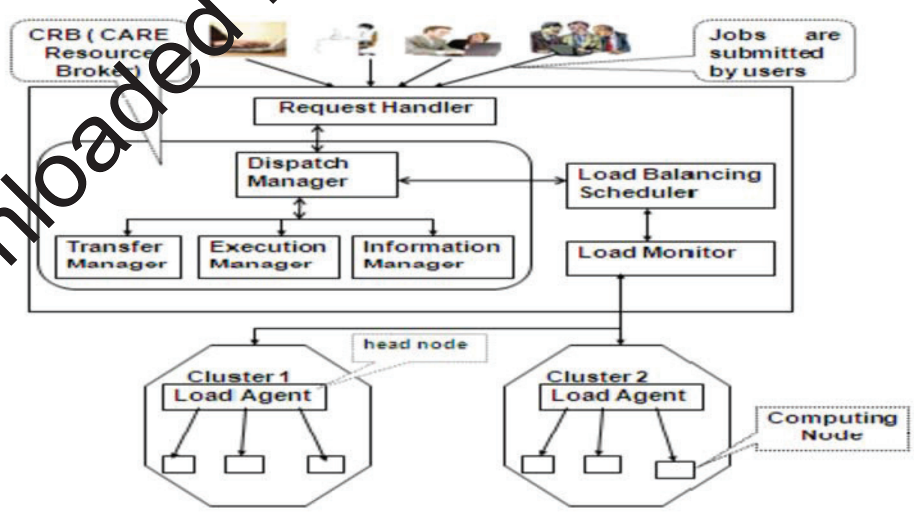


Fig 1: Structure of Load Balancing Grid scheduler

This scheduler will perform the load balancing and job migration, by exploiting the information gathered from load monitor and information manager, along with the topology and load cost. It will allocate the job to the resource which is having less topology cost to reach and less load cost. The Load Monitor contains the information about all the load agents, and prepares system load information about all the resource. This load information is automatically updated from the load agent when there is any change in the resource load by using event notification approach. The structure of the Load Monitor is shown in Fig 2. Finally the load information is given to the LBS for actual Load Balancing and job migration. The Load Agent service acts as a server and a copy of load agent service has to run on all resource or cluster where the users want to run their applications. Load agent provides system load information such as job queue length, CPU speed and the number of node count available in the resource.

The information manager will query the Monitoring and Discovery Service (MDS) and sends the host information to the LBS. Based on the monitoring interval it keeps track of the host status and updates the host information to the LBS. If any new resources are added those information are also updated periodically [18]. The Transfer Manager is invoked by the dispatch manager with the job-id and the matching resource-id as input. Once it is invoked, the transfer manager creates a remote directory for the given path name in user input. Transfer manager gives the permission rights for the execution of given job in the remote directory. Once this process is over, it informs the dispatch manager through messages.

The execution manager is invoked by the dispatch manager when the transfer manager completed the creation of directory in the remote host. The dispatch manager will dispatch the job for execution. Execution manager will keeps updating the job status to the scheduler. Finally it reports the completion or failure of job to the scheduler.

Before representing the exact problem statement, we first describe the notations and terminology that are used throughout the paper as
shown in Table 1.

Table 1

List of notations and terminology

M	Number of heterogeneous resources (R1,R2, ..., RM)
N	Number of jobs to be processed
Di	Delay to reach the resource Ri
TFi	Topology Factor of the resource Ri (Number of hop to reach the resource)
TCi	Topology cost of the resource
JUi	Job unit of the task or job Ji (size of job)
λi (T)	Estimated arrival rate for the resource Ri at time T
μi (T)	Estimated service rate for the resource Ri at time T
α	Boundary value estimation factor
β	Threshold value estimation factor
LT	Threshold load value of the resource
Llow	Resource is of low load
Lhigh	Resource is of high load
Lmoderate	Resource is of moderate load
Li (T)	Estimated load of the resource Ri at time T
TLR(T)	Total load of all the resource R = (R1,R2, R3..., RM)
ALR(T)	Average load of all the resource R
LUBR	Load upper boundary for the resource R
LLBR	Load lower boundary for the resource R

We will now introduce certain key performance metrics considered in this paper such as Topology and Load cost, regarding the Grid Scheduling and Load Balancing in the real Grid Environment.

3.3 Estimation of Topology Cost

Topology is regarded as one of the outmost important factors since it could strongly affect the Grid performance [12]. In our proposed approach, LBS will take the decision to submit the job to best fit resource, which is having less topology cost to reach, by exploiting topology information. Here the topology information, such as delay and number of hop count is gathered by using the Network Weather Service Tool. The topology cost is calculated as follows,

$TC_i = 2 * D_i * TF_i$ (1)

Here the delay D_i represents the latency or time to reach the particular resource R_i. And the topology factor TF_i represents the number of hop to reach the particular cluster or resource R_i. The impact of considering topology cost will reduce the request and response time (elapse time) of job [11].

3.4 Estimation of Load Cost

One of the key performance metrics considered for load balancing is Load Cost. The major impact of considering this metrics will help in choosing the best resource. For the estimation of load cost, the resource information such as CPU speed, free memory and number of node count is pulled by using the Load Agent. The important factor which needs to be the part of the scheduling and load balancing algorithm is the Load Cost. First we calculate the job unit JU_i of the job J_i as follows,

$JU_i = NC (J_i)$ (2)

Where NC denotes the number of node count required by the job J_i. Then calculate the estimated arrival rate $\lambda_i (T)$ of the resource R_i at the time T as follows,

$\lambda_i (T) = \Sigma (JU_1,JU_2,JU_3,\ldots,JU_N)$ (3)

where $JU_1,JU_2,JU_3, \ldots, JU_N$ represents the number of jobs units present in the queue of the resource R_i [21]. And the Estimated service rate $\mu_i (T)$ of the resource at the time T is expressed as,

$\mu_i (T) = \Sigma (C_1,C_2,C_3,\ldots,C_k) * \text{CPU speed}_i$ (4)

Where $C_1,C_2,C_3, \ldots,C_k$ indicates the number of computing node present in the resource R_i, and the service rate vary with respect to CPU speed as well as the node count of the resource [20]. Finally the load cost $L_i(T)$ of the resource R_i is estimated by taking the ratio of the estimated arrival rate to the estimated service rate using equation (3) and (4) [8], [9], [18].

$L_i (T) = \lambda_i (T) / \mu_i (T)$ (5)

The impact of considering Load Cost, will helps in choosing the best resource which is least loaded, reduces the waiting time of the job and also avoid the overloading of resources.

4 LOAD BALANCING SCHEDULER ALGORITHM

In this paper, LBS algorithm is proposed which performs the job to resource matching in the dynamically varying grid environment. The job information is placed in the Job Pool and the resource information is placed in the Resource Pool.

Table 2

Job information present in Job Pool

Job Id	Required Free Memory	Required CPU speed	Required Node count
1	80	1.4	1
2	60	2.0	7
3	60	3.0	7
4	20	1.4	1
5	20	3.0	3

In the Job Pool, the job information such as job id, required free memory, required CPU speed and required number of node count is present as shown in Table 2. Based on the job information, job is classified as data intensive and computation intensive jobs. Similarly in the Resource Pool, the resource and topology related information such as resource id, available free memory, CPU speed, delay to reach the resource, number of hop count to reach the resource and the number of task remaining in resource or cluster queue are present as shown in Table 3.

Table 3

Resource information present in Resource Pool

Resource						
Id	Delay to Reach	No. of Hope	Available Free Memory	CPU Speed	No. of Node Count	No. of Jobs Unit in the Queue
1	0.160	3	3000	1.4	5	10
2	0.305	5	8000	2.0	15	30
3	0.160	3	2000	3.0	5	10
4	0.305	5	9000	3.2	15	50
5	0.243	7	5000	1.4	5	20

In the LBGS, matchmaking is a process in which, each job in the Job Pool is matched with the set of resource available in the Resource Pool. Assume there are 'N' number of jobs and 'M' number of resource available in Job Pool and Resource Pool respectively.

The Load Balancing algorithm such as LBS works as shown in Algorithm 1.This algorithm will takes the job and resource information as input and gives the output as best resource for each job [10], [18].

Algorithm 1

Load balancing Scheduler (LBS) Algorithm

```
Get the list of jobs present in Job Pool
for ( all the jobs )
{
Identify the type of Job
If ( resource = = 1 )
{
Submit the Job
}
else if (Job is Data intensive)
{
Identify the set of matched resource
for ( Set of matched resource )
{
Find the Topology Cost for all the nodes
Choose resource having minimum Topology Cost
if ( less Topology Cost resource = =1 )
{
Submit the Job
}
else
{
Calculate the Load Cost for all the nodes
Choose resource having minimum Load Cost
Then submit the Job
}
}
}
else if ( Job is computation intensive )
{
Identify the set of matched resource
for ( set of matched resource )
{
Find the Load cost for all the nodes
Choose resource having minimum Load cost
if (minimum Load cost resource = =1)
{
Submit the Job
}
else
{
Calculate the Topology cost for all the nodes
Choose resource having minimum Topology
Then submit the Job
}
}
}
```

Else (Job is both data and computation intensive)
{
Identify the set of matched resource
for (Set of matched resource)
{
Calculate topology and load cost for all the nodes
Calculate the total cost for all the nodes
Total cost = Topology cost + Load cost
Choose resource having minimum Total cost
Then submit the Job
}
}
}

In this algorithm, classification is done first such as whether data intensive or computation intensive. If the job requires more free memory but require less computation node means, then the job isconsidered to be a Data intensive job. Suppose, if the job requires more computation node, but requires only less amount of free memory means, then the job is considered to be a computation intensive job. Otherwise the job is considered to be both data and computation intensive. Based on its classification it calls the other three methods for its computation and best resource selection. Suppose, if the job is data intensive, it compute the topology cost as the foremost process for the choosing the best resource. Here the Load cost is calculated only on demand. In case, if the job is computation intensive then it compute the load cost and choose the best resource for submitting the job. The topology cost is computed only if too many resources were matched with minimum load cost range. Suppose if the job doesn't falls under either data intensive or computation intensive, it concludes that the job is both data and computation intensive, and it compute total cost for all the nodes and finally it chooses the resource having minimum of total cost.

5 LOAD BALANCER AND JOB MIGRATION ALGORITHM

Of course, there may be lot of scheduling algorithms were proposed by different authors, which works well in the specified environment. In some cases, too many numbers of jobs may have the same required, which will definitely leads to overloading of particular resource. So the application level load balancing is needed for the individual parallel jobs. In the proposed model, many load balancing policies such as Information, Selection, Location and Triggering policy is considered. And the Triggering policy determines the appropriate time period to start a load balancing operation.

Suppose, if the Load Balancing is done very often means, then it leads to the performance degradation and drawback to system. So this load Balancer algorithm can be initiated based on two approaches by using the triggering policy. In the first approach, boundary value is used when no jobs arrived to the meta-scheduler for the specified time interval. In the second approach threshold value is used when the Load in the resource exceeds the particular threshold value.

The Load Balancer (LB) algorithm using boundary value approach will works as shown in Algorithm 2. This algorithm will get the resource load information from the Load Monitor service, which is present in the meta-scheduler. First this LB algorithm calculates the total load of the resource TLR(T) by adding all the resource load information at time T as follows,

$TLR(T) = \Sigma\ L1\ (T),\ L2\ (T),\ldots,\ Lm\ (T)$ (6)

Where L1, L2,L3,...,Lm represents the Load of the resource R1,R2,R3,..., Rm respectively. Then it compute the average of load resource ALR(T) at the time T, by taking the ratio of the TLR(T) to the total number of resource available 'm' using the equation (6).

$ALR(T) = TLR(T) / m$ (7)

Then set the upper and lower boundary by using the equation (7) and the parameter α. The load upper boundary of the resource R is expressed as,

$LUBR = ALR(T) + \alpha$ (8)

And the load lower boundary of the resource R is expressed as,

$LLBR = ALR(T) - \alpha$ (9)

where α is the parameter denotes the boundary value estimation factor and its value is based on the multiprocessor system [5].

Algorithm 2

Load Balancer (LB) Algorithm using Boundary value

```
Get the Load information for all resource using Load Monitor service
Calculate Load Cost for all resource
Calculate the total Load Cost TLR(T)
Compute average load cost ALR(T)= total load cost/n
Set the upper boundary as LUBR = ALR(T) + &
Set the lower boundary as LLBR = ALR(T) - &
for ( all the resource )
{
if ( resource Load Cost > LUBR)
{
Node is overloaded
Use Algorithm 4
}
else if ( resource load cost < LLBR )
{
Node is least loaded
Get the Job for execution
}
else
{
Resource is moderately load
No need for job migration here
}
}
```

Resource which have the load greater then LUBR is said to be overloaded and the resource which have the load lesser then LLBR is said to least loaded by exploiting equation (8), (9). A resource is moderate means the resource load is in the state between LUBR and LLBR [7]. Suppose, if the load information exceeds the boundary value, then the load balancing policies such as Selection, Information and Location policy are considered to migrate the job to other resources which are below the boundary value. If the resource is overloaded means Job Migration (JR) algorithm is used which works as shown in Algorithm 4.

The Load Balancer (LB) algorithm using threshold value approach will works as shown in Algorithm 3. This algorithm will get the resource load information from the Load Monitor service, which is present in the meta-scheduler. First this LB algorithm assigns the threshold value LT to the resource as follows,

LT = β (10)

where β is the parameter denotes the threshold value estimation factor and its value is based on the resource service rate [5]. If the load Li of the resource Ri falls below the threshold load LT, then the resource load is assigned as,

Llow = Li (11)

Suppose, if the load Li of the resource Ri is greater than the threshold load LT, then the resource load is assigned as,

Lhigh = Li (12)

In case, if the load Li of the resource Ri is equal to the threshold load LT, then the resource load is assigned as,

Lmoderate = Li (13)

Where Li denotes the load of the resource Ri.

Algorithm 3

Load Balancer (LB) Algorithm using Threshold value

```
Get the Load information of all resource using Load Monitor service
Calculate Load Cost for all resource
Set the Threshold load value of the resource as LT
if ( the load Li of the resource < LT )
{
Set Ri as low load resource Llow
}
else if (the load Li of the resource > LT )
{
Set Ri as high load resource Lhigh
}
else
{
Set Ri as moderate load resource Lmoderate
}
for ( all the resource )
{
if ( the load Li of the resource Ri == Lhigh )
{
Node is overloaded
Use Algorithm 4
}
else if ( the load Li of the resource Ri == Llow)
{
Node is least loaded
Get the Job for execution
```

```
}
else
{
Resource is moderately load
No need for job migration here
}
}
```

Using equation (11), (12) and (13) identify the overloaded resource and then finally use load balancing policies such as Selection, Information and Location policy to migrate the job to other least loaded resources which are below the threshold value using the Job Migration (JR) algorithm as shown in Algorithm 4.

Algorithm 4

Job Migration Algorithm

```
Get the list of overloaded resource
Get the list of least loaded resource
for ( all the overloaded resource)
{
While (overloaded resource Load Cost > LUBR)
{
Take the one Job from Resource Queue
for ( all least loaded resource )
{
Find the Load cost for all the nodes
Choose resource having minimum Load Cost
if (minimum Load cost resource = =1)
{
Then Migrate the Job to the resource
}
else
{
Find the Topology cost for all the nodes
Choose resource having less Topology Cost
Then Migrate the Job to the resource
}
}
```

Once the job migration algorithm is executed successfully means, we can realise the load balancing and also the work load is get distributed among all the resource. As a result of load balancing and job migration, waiting time of the job is reduced, maximize the utilization of the resource which is least loaded and increases the overall performance of the resource.

6 EXPERIMENTAL RESULT AND PERFORMANCE EVALUATION

In this result phase the main focus is to show the result, as the proposed scheduler performs well, when comparing to the normal scheduler by considering all the challenging issues. We have simulated the result by exploiting ten resources and hundreds of jobs. The result of both the scheduling and LBGS is compared and the performance measure is also represented as graph. The performance of the proposed LBGS is more efficient in elapse time of job, with respect to delay as shown in Fig 2.

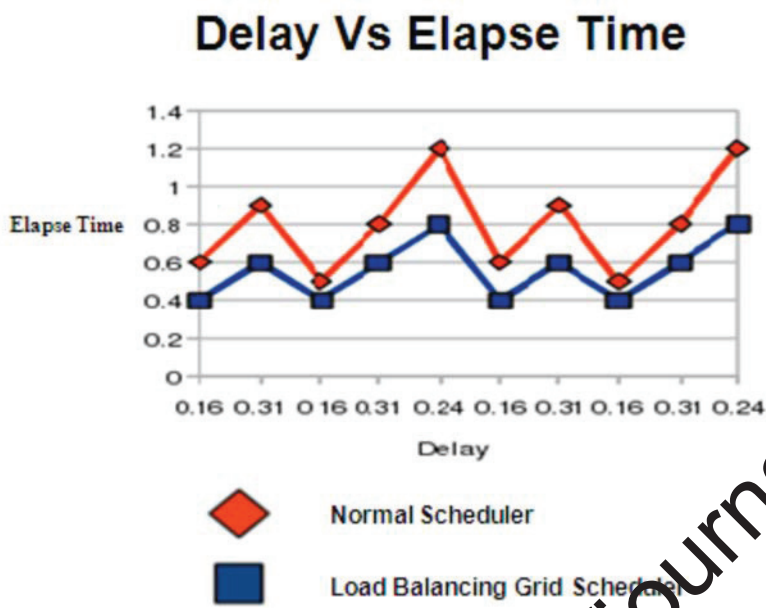
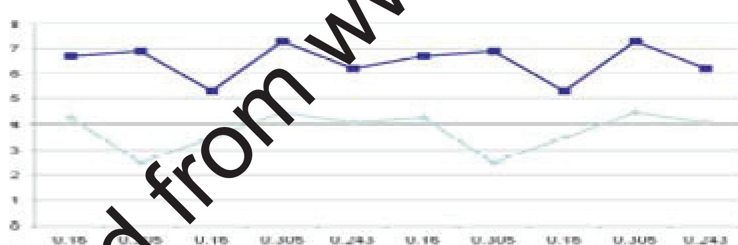


Fig 2: Performance evaluation of Normal Scheduler Vs Load Balancing Grid Scheduler with respect to delay and elapse time.

Load Vs Waiting time of Job



Waiting Time of Job
Load
Normal Scheduler
Load Balancing Grid Scheduler

Fig 3: Performance evaluation of Normal Scheduler Vs LoadBalancing Grid Scheduler with respect to load and waiting time of job.

Similarly, the performance evaluation of this scheduler is done with respect to the waiting time of job and the load of the resource as shown in the Fig 3. Since the performance of the system vary with respect to delay, as well as the load. So, the parameter such as delay, load, waiting time and elapse time is considered for the performance evaluation.

7 CONCLUSION AND FUTURE WORK

ur proposed model exhibits different features like topology aware and dynamic load balancing during the job firing to list of matched resource. As far as the deployment issues are concern, it providessupport for multiple users, flexible and extensible architecture. Hence the meta-scheduler is developedwith all the features, in which the load balancing is realized during the scheduling. We have evaluated the proposed LBS with the simulation and traces from real time Grid environment in CARE laboratory. The result obtained with performance evaluation, can balance the load, decreases the elapse time and increase the utilization of the resource, which are idle or least loaded. So the obtained result shows, the proposed LBGS model with load balancing performs better, then normal scheduling in terms of delay and load.

In feature, we are working towards Load Balancing and job migration between the metascheduler in the real Grid Environment.

8 ACKNOWLEDGEMENTS

We wish to express our thanks to Centre for Advance Computing Research and Education (CARE) team members in the research laboratory, for their helpful and constructive suggestion, which considerably improve the quality of the paper. This paper has been supported by the CARE, for implementing the Grid Scheduling and Load Balancing algorithm in the real Grid Environment.

9 REFERENCES

[1] I. Foster and C. Kesselman, “The Grid: Blueprint for a Future Computing Infrastructure”, Morgan Kaufmann, 1999.

[2] I. Foster, and C. Kesselman, “Globus: a meta computing infrastructure toolkit,” International Journal of High Performance Computing Applications, vol. 2, pp. 115– 128,1997.

[3] George Coulouris, Jean Dolimore and Tim Kindberg, “Distributed Systems: Concepts and Design”, Addison Wesley Longman, 1994.

[4] Thamarai Selvi Somasundaram, Balachandar R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajandar, R. Rajiv, G. Kannan, G. Rajesh Britto, E. Mahendran and B. Madusudhanan, “CARE Resource Broker: A framework for scheduling and supporting virtual resource management”, Journal: Future Generation Computer System, 2010.

[5] HAN Xiangchun1, CHEN Duanjun1, CHEN Jing, “One centralized scheduling pattern for dynamic load balance in grid”, IEEE International Forum on Information Technology and Applications, 2009.

[6] Ivan Rodero, Francesc Guim, Julita Corbalan, “Evaluation of Coordinated Grid Scheduling Strategies”, 11th IEEE International Conference on High Performance Computing and Communications, 2009.

[7] Bin Lu and Hongbin Zhang “Grid Load Balancing Scheduling Algorithm Based on Statistics Thinking”, 9th IEEE International Conference for Young Computer Scientists, 2008.

[8] Naveen Kumar Gondhi and Dr. Durgesh Pant, “An Evolutionary Approach for Scalable Load Balancing in Cluster Computing”, IEEE International Advance Computing Conference, 2009.

[9] Ruchir Shah, Bhardwaj Veeravalli and Manoj Mistra, “On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments”, IEEE Transactions on Parallel and Distributed Systems, December 2007.

[10] Ashiq Anjum, Richard McClatchey, Arshad Ali and Ian Willers, "Bulk Scheduling With the DIANA Scheduler", IEEE Transactions On Nuclear Science, December 2006.

[11] Ioannis Psoroulas, Ioannis Anagnostopoulos, Vassili Loumos And Eleftherios Kayafas, "A Study Of The Parameters Concerning Load Balancing Algorithms", Ijcsns International Journal Of Computer Science And Network Security, April 2007.

[12] Gregory A.Koenig And Laxmikant V.Kale, "Optimizing Distributed Application Performance Using Dynamic Grid Topology - Aware Load Balancing", Department Of Computer Science, University Of Illinois At Urbana- Champaign, 2007 IEEE.

[13] Belabbas Yagoubi, Hadj Tayeb Lila and Halima Si Moussa, "Load Balancing in Grid Computing", Asian Journals of Information Technology, 2006.

[14] Quan Liu and Yeqing Liao, "Grouping based Fine grained Job Scheduling in Grid Computing", IEEE First International Workshop on Education Technology and Computer Science, 2009.

[15] Yigang, Hui Dong and Ziran Zhang, "An Open Source Toolkit for Grid Resource Scheduling in Digital Library", IEEE Ninth International Conference on Hybrid Intelligent Systems, 2009.

[16] SHI Lei and XU Hui-hui, "Grid resource Scheduling Algorithm Based on QoS Guide GA", IEEE, 2009.

[17] Xiu-mei Wen, Wei Zhao and Fan-xing Meng, "Research of Grid Scheduling Algorithm based on P2P Grid Model", IEEE International Conference on Electronic commerce and Business Intelligence, 2009.

[18] Yi Hu and Bin Gong, "Multi-objective optimization approach using a CE-ACO inspired strategy to improve Grid Job Scheduling", IEEE Fourth China Grid Annual Conference", 2009.

[19] Kuo-Chan Huang, "On Effects of Resource Fragmentation on Job Scheduling Performance in Computing Grids", IEEE 10th International Symposium on Pervasive Systems, Algorithms and Networks, 2009.

[20] Menno Dobber, Rob Van Der Mei And Ger Koole, "Dynamic Load Balancing And Job Replication In A Global-Scale Grid Environment: A Comparison", IEEE Transactions On Parallel And Distributed Systems, February 2009.

[21] Daniel Grosu and Anthony T. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed System", IEEE Transactions on Systems, man and Cybernetics, February 2004.